Université de M'sila

FACULTÉ DES MATHÉMATIQUES ET INFORMATIQUES

Département de Mathématiques

Mémoire de fin d'étude

Présenté pour l'obtention du diplôme de **Master**

Domaine: Mathématiques et Informatique

Filière: Mathématiques

Spécialité: Mathématiques Discrètes

Par

DJEMIAT Sara

Sujet

Des problèmes polynomiaux: Problème d'ordonnancement d'atelier

Président

Hocine Belouadah

Promotion: 2014/2015

Remerciements

Tout d'abord, je remercie ALLAH le Tout Puissant de m'avoir donné la santé, le courage, et la patience pour réaliser ce travail.

Je tiens à remercier mes promoteurs: Mr le Professeur **Hocine BELOUADAH** pour la confiance qu'il m'a témoignée en me proposant ce sujet, ses encouragements et sa patience.

Mr **Abderrahim GUERNA** qui ma aussi aider avec ses conseils, les discussions scientifiques qu'il a su générer, ses remarques et ses suggestions qui m'ont permis de finaliser ce modeste travail. Je souhaite leurs transmettent ma reconnaissance et ma plus profonde gratitude.

Meriem YOUSFI ensuite; sa gentillesse sa patience, son aide, ses précieux conseils, et la confiance qu'elle m'a accordée m'ont accompagnée tout au long de cette étude.

Je remercie aussi tous les membres du Jury pour l'honneur qu'ils m'ont fait, en acceptant de juger ce modeste travail.

Je ne peux pas clôturer mes remerciements sans se retourner vers les êtres qui me sont les plus chers; ma famille qui ont eu un rôle essentiel et continu dans ma réussite.

Merci

Table des matières

Introduction

Parmi les problèmes rencontrés par le chercheur et l'ingénieur, les problèmes d'optimisation occupent à notre époque une place de choix. Formuler les problèmes d'optimisation et tenter de les résoudre représente l'objectif principal de nombreux chercheurs.

Comprendre, analyser et formuler un problème d'optimisation nécessitent d'abord une définition des paramètres, des variables, de l'espace de recherche ainsi que des fonctions à optimiser.

Une fois la (ou les) fonction(s) à optimiser définie(s), une méthode adaptée pour la résolution du problème posé est choisie.

A ce niveau, la taille et la complexité du problème entrent en compte pour le choix de la méthode d'optimisation. Si le problème est de petite taille et de complexité réduite, la mise en œuvre d'une méthode exacte peut suffir et aboutir à une solution optimale.

Dans le cas de problèmes de tailles importantes, les méthodes approchées constituent le moyen le plus efficace de se rapprocher le plus possible de la solution optimale.

Les problèmes d'ordonnancement d'atelier constituent sûrement pour les entreprises une des difficultés importantes et leurs systèmes de gestion et de pilotage de la production. En effet, c'est à ce niveau que doivent être prises en compte les caractéristiques réelles multiples et complexes des ateliers. Dans ce type d'ordonnancement, les ressources sont généralement des machines, et chaque tâche à ordonnancer concerne un produit ou un lot de produits à fabriquer en respectant les gammes de fabrication.

Dans le premier chapitre de ce mémoire, le contexte général de l'ordonnancement et les problèmes d'ordonnancement est décrit afin de replacer le sujet dans son cadre général. On commence par une représentation des notions de base concernant la production et la gestion de production. La suite du chapitre est consacrée au cadrage de la problématique générale

de l'ordonnancement par présentation des points essentielles: les tâches, les ressources, les contraintes, les critères. Puis une vue d'ensemble sur les problèmes d'ordonnancement des systèmes de production et sur leurs complexités.

Le deuxième chapitre, vise à présenter les différentes composantes de l'ordonnancement d'atelier. Là nous avons examiné les différents problèmes à une machine et machines parallèles. Ainsi les classifications des problèmes d'ordonnancement dans un atelier peut s'opérer selon le nombre de machines et leurs ordres d'utilisation pour fabriquer un produit, qui dépend de la nature de l'atelier considéré. Un atelier est caractérisé par le nombre de machines qu'il contient et par son type Flow-shop, Job-shop et Open-shop pouvant les caractériser sont introduits. Différentes représentations possibles des problèmes d'ordonnancement d'atelier sont par la suite proposées. En fait on s'est concentré sur les problèmes faciles (polynomiaux) et leurs algorithmes de la solution que se soit les problèmes à une ressource ou à multiples ressources.

Dans le troisième chapitre, une étude de cas est prise. C'est la comparaison numérique de deux heuristiques $SWPT$ et EDD pour la détermination d'une solution locale au problème d'ordonnancement successivement où l'objectif est du minimise la somme des retard pondères ($NP\ difficile$). L'algorithme général est programmé en utilisant le langage Java.

Chapitre 1

Éléments Fondamentaux

1.1 Introduction

Les problèmes d'optimisation combinatoire abordés dans la littérature sont la plupart du temps issus de situations concrètes de l'industrie ou de service. Ils représentent une vue simplifiée d'une situation, et leurs résolutions aide à la prise de décision [6].

Ce chapitre se penche sur l'étude d'un des problèmes combinatoires, à savoir les problèmes d'ordonnancement en général.

1.2 Notations générales

Soient $i = \{1, 2, ..., n\}$ l'ensemble des n tâches à ordonnancer et $j = \{1, 2, ..., m\}$ l'ensemble des m machines. Le tableau suivant résume les différentes notations utilisées dans les problèmes d'ordonnancement d'atelier.

M	Nombre de machines.
N	Nombre de tâches.
i	Indice de tâche.
j	Indice de machine.
$p_{i,j}$	Durée de la tâche i sur la machine j.
r_i	Date de disponibilité de la tâche i (date minimale de début d'exécution).
d_i	Date d'achèvement souhaitée de la tâche i.
w_i	Poids de la tâche i.
c_i	Date de fin de la tâche i.
C_i	Date de fin d'exécution de la tâche i.
F_i	Temps de présence dans le système de la tâche $i : F_i = C_i - r_i^2$.
L_i	Retard algébrique de la tâche $i : L_i = C_i - d_i$.
T_i	Retard vrai de la tâche $i : T_i = max(C_i - d_{i,}0)$.
U_i	Indicateur de retard de la tâche $i : U_i = 1, si\ T_i > 0, U_i = 0$, sinon.

Tableau 1.1: les notations utilisées dans les problèmes d'ordonnancement d'atelier.

- Les données d'un problème d'ordonnancement sont:
  - Un ensemble de **tâches**.
  - Un environnement de **ressources** pour effectuer les tâches.
  - Des **contraintes** sur les tâches et les ressources.
  - Un **critère d'optimisation;**

On veut déterminer les dates d'exécution des tâches de telle manière à optimiser un critère.

1.3 Les tâches

Une tâche i est une entité élémentaire de travail (opération ou ensemble d'opérations), décrite par les trois caractéristiques suivantes:

- Caractéristique d'époques: une tâche doit avoir des limites chronologiques bien définies:

$$\begin{cases} r_i\text{: date de disponibilité;} \\ t_i\text{: date de début de la tâche;} \\ c_i\text{: date de fin de la tâche.} \end{cases}$$

- Caractéristique de durée:

  ($p_i = c_i - t_i$ durée de la tâche) .

- Caractéristique de moyens:

  Il s'agit de divers moyens (matériels, personnel, fournitures, monnaie, ..., etc.) qui sont nécessaires à la réalisation.

Caractéristiques des tâches

– Modes d'exécution des tâches:

Sans préemption: une tâche commencée doit être exécutée jusque la fin.

Avec préemption: tâches morcelables, une tâche ou une opération peut être interrompue et recommencée plus tard sur la même machine ou sur une autre [16].

1.4 Les ressources

Une **ressource** k est un moyen technique ou humain requis pour la réalisation d'une tâche et disponible en quantité limitée, sa **capacité** w_k.

On distingue plusieurs types de ressources. Une ressource est renouvelable si après avoir été utilisée par une ou plusieurs tâches, elle est à nouveau disponible en même quantité (les hommes, les machines, l'espace, ..., etc.); la quantité de ressources utilisée à chaque instant est limitée. Dans le cas contraire, elle est consommable (matière première, budget, etc.); la consommation globale (ou cumul) au cours du temps est limitée.

On distingue par ailleurs:

-les ressources **disjonctives** (ou non partageables) qui ne peuvent exécuter qu'une tâche à la fois (machine-outil, robot manipulateur).

-les ressources **cumulatives** (ou partageables) qui peuvent être utilisées par plusieurs tâches simultanément (équipes d'ouvriers, poste de travail) [5] [15].

La nature des ressources prises en considération permet de dresser une typologie des problèmes d'ordonnancement qui est la suivante:

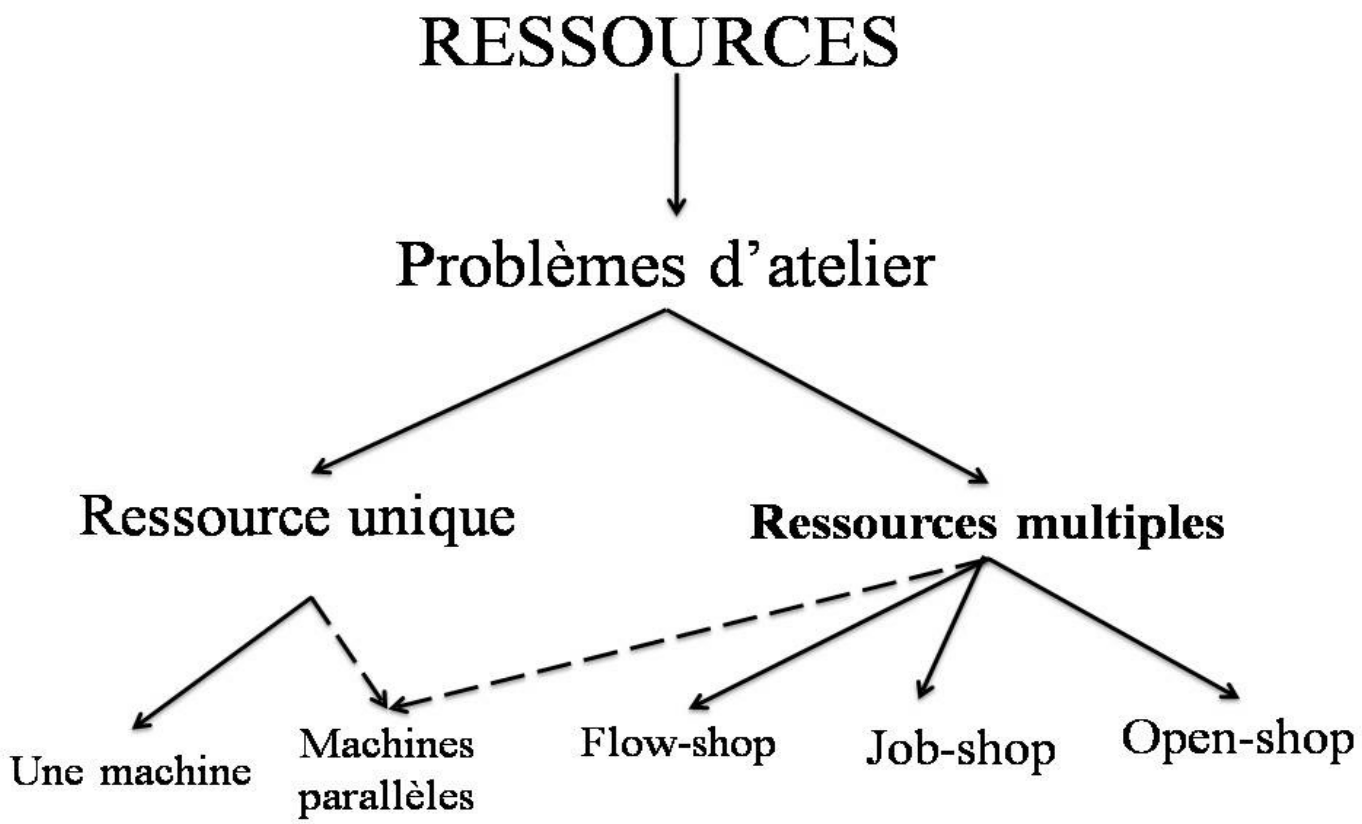

Figure 1.1: typologie par les ressources des problèmes d'ordonnancement.

1.5 Les contraintes

Une contrainte est une restriction sur les valeurs que peuvent prendre une ou plusieurs variables de décision sur le temps (variable d'ordonnancement) ou bien sur les ressources [13].

Selon [1], les contraintes auxquelles sont soumises les diverses tâches qui concourent à la réalisation du projet sont de divers types. On distingue:

Contraintes temporelles

Les contraintes temporelles intègrent en général:

- Les contraintes de temps alloué.
- Les contraintes de calendrier liées au respect d'horaires de travail, etc.

Contraintes de ressources

Contraintes disjonctives Deux tâches i et j sont en disjonction si elles ne peuvent être exécutées simultanément.

Contraintes cumulatives

- C'est une généralisation des contraintes disjonctives.

1.6 Les critères

L'approche par optimisation suppose que les solutions candidates à un problème puissent être ordonnées de manière rationnelle selon un ou plusieurs critères d'évaluation numériques permettant d'apprécier la qualité des solutions. On cherchera à minimiser ou maximiser de tels critères. On note par exemple ceux:

- Liés au temps [11]:
  - Le temps total d'exécution ou le temps moyen d'achèvement d'un ensemble de tâches.
  - Les retards (maximum, moyen, somme nombre, ..., etc.) par rapport aux dates limites fixées.
- Liés aux ressources:

  - La quantité-maximale, moyenne ou pondérée de ressource nécessaire pour réaliser un ensemble de tâches.
  - La charge de chaque ressource.

- Liés aux coûts de lancement, de production, de transport, de stockage, ..., etc. Mais aussi aux revenus, aux retours d'investissements.

Critère d'optimisation

Il faut ordonnancer les tâches de façon à optimiser un certain objectif qui sera l'un des trois types suivants:

- L'utilisation efficace des ressources.
- Le délai d'exécution des tâches soit faible que possible. C'est le critère le plus employé dans le problème central de l'ordonnancement.
- Le respect des dates d'exécution des tâches décrites à l'avance.

1.7 Complexité des problèmes

Distinction entre problème de décision et problème d'optimisation;

- Un problème de décision est un problème pour lequel une solution est soit "oui" soit "non".
- Un problème d'optimisation est un problème pour lequel on doit chercher à déterminer une solution qui optimise un critère.
- A chaque problème d'optimisation on peut associer un problème de décision.

Problème de décision:

Est un problème dont les résultats ne peuvent prendre que l'une des deux valeurs: *oui* ou *non*.

La classe P:

Un problème est dit polynomial s'il existe un algorithme de complexité polynomiale permettant de répondre à la question posée dans ce problème, quelle que soit la donnée de celui-ci.

La classe P est l'ensemble de tous les problèmes de reconnaissance polynomiaux [3].

La classe NP:

Un problème de décision appartient à la classe NP s'il peut être résolu par un algorithme non déterministe polynomial.

Parmi la classe de problème NP, on distingue deux grandes classes:

- La classe des problèmes polynomiaux (la classe P) et la classe des problèmes NPComplet (la classe NPC).

La classe NP-Complet:

Un problème de décision est dit $NP - Complet$ si tout problème de la classe NP peut se rendre polynomialement à lui.

- Tous les problèmes $NP - difficiles$ ne sont pas de difficulté identique. On rencontre par exemple des problèmes qui ne peuvent pas être résolus en temps polynomial avec un codage binaire, mais qui peuvent l'être avec un codage unaire, ces problèmes sont dits NP-difficiles au sens ordinaire ou simplement $NP - difficiles$, les algorithmes pour cette classe de problèmes sont appelés pseudo-polynomiaux, pour d'autres problèmes, on peut ne pas trouver d'algorithme polynomial, quelque soit le codage, ceux-là sont dits $NP - difficiles$ au sens fort.

Définition 1.7.1 *On considère deux problèmes de décisions Q et R. On dit que Q se réduit polynomialement à R, et on note $Q \propto R$, s'il existe une fonction polynomiale f qui transforme toute instance de Q en une instance de R telle que x est une réponse "oui" de Q si et seulement si $f(x)$ est une réponse "oui" de R.*

Définition 1.7.2 *Un problème R est $NP - complet$ ssi $R \in NP$ et $\forall Q \in NP, \exists \propto$ telle que $Q \propto R$* [12].

1.8 Complexité des algorithmes

- Un même problème peut généralement être résolu par plusieurs algorithmes

$\Longrightarrow$ Il faut comparer entre ces algorithmes.

- La comparaison se base sur **le temps de calcul** et sur **l'espace mémoire** requis par l'algorithme.

Définition 1.8.1 *On appelle, complexité en temps d'un algorithme dans le pire cas, la fonction $f(n)$ qui donne une borne supérieure du nombre d'opérations élémentaires effectuées par l'algorithme lorsque la taille de l'entrée est n.*

1.9 Conclusion

L'ordonnancement est généralement décrit comme une fonction particulière de décision au sein d'un système de gestion du travail concernant la production de bien, d'ouvrages ou de services.

La majorité des problèmes d'ordonnancement sont NP-complets, ça veut dire que, dans la pratique, la complexité croît exponentiellement avec le nombre de tâches et de ressources. Il n'est donc envisageable de résoudre, de tels problèmes avec les méthodes exactes. C'est pour cela qu'il faut développer des heuristiques dont l'objectif est de fournir des solutions aussi proches que possible de la solution exacte en un temps raisonnable.

Chapitre 2

Ordonnancement d'atelier

2.1 Introduction

Dans les problèmes d'ordonnancement d'atelier, les ressources sont des machines. On considérera successivement les problèmes à machine unique et les problèmes multi-machines. Dans le premier cas, les tâches se réduisent à une seule opération; il faut alors trouver une séquence des tâches sur la machine unique. Dans l'autre, on distinguera tout d'abord les problèmes à machines parallèles identiques, qui conservent l'hypothèse de tâches à opération unique et pour lesquels la solution est un ensemble de séquences, une par machine. On étudiera ensuite les problèmes d'ateliers multi-machines où les tâches compostent plusieurs opérations, chacune nécessitant une machine particulière disponible en exemplaire unique. Ce cas recouvre lui-même trois types de problèmes, selon que l'ordre des opérations composant un même tâche est fixé et commun à tous les tâches (Flow-shop), qu'il est fixé mais propre à chaque tâche (Job-shop), où enfin qu'il est indéterminé (Open-shop).

2.2 Modèle de base

2.2.1 Données, variables et contraintes

- L'atelier est constitué de m machines différentes $M_j, j \in \{1, ..., m\}$.
- n **travaux** ou "tâches" doivent être réalisés J_i, $i \in \{1, ..., n\}$; ils sont tous exécutables à $t = 0$.

- Chaque travail J_i est composé de n_i opérations.
- L'opération j du travail i est note (i, j) ;
  - Elle utilise la machine m_{ij}.
  - Sans interruption pendant toute sa durée p_{ij} pour l'opération i du tâche (le temps de traitement).
  - Le respect de la gamme de fabrication impose:

$(i, 1) \prec (i, 2) \prec ... \prec (i, j) \prec ... \prec (i, n_i)$ où le signe '$\prec$'signifie précède.

- Une date de fin souhaitée d_i (due date).
- Un poids relatif w_i (weight).

2.2.2 L'objectif du problème d'ordonnancement

L'objectif du problème d'ordonnancement est de fixer les dates de début t_{ij} des opérations (i, j). Pour cela, les ressources étant disjonctives, il faut déterminer **l'ordre de passage** de l'ensemble des tâches sur chaque machine, ou **séquence**. À partir de ces séquences, plusieurs **ordonnancements admissibles** localisant de manière absolue les opérations dans le temps- peuvent être obtenus selon les critères que l'on cherche à optimiser.

2.2.3 Critères

En ordonnancement d'atelier, les indicateurs de performances varient essentiellement en fonction des temps d'attente que subissent les produits entre deux opérations, s'expriment en fonction des dates de fin des tâches C_i (**completion times**). Pour un tâche i, Les critères sont généralement exprimés en fonction des mesures suivantes [2]:

- Le retard algébrique (relatif) $L_i = C_i - d_i$ (**lateness**).
- Le retard absolu $T_i = \max(0, C_i - d_i)$ (**tardiness**).
- L'avance $E_i = max(0, d_i - C_i)$ (**earliness**).

- La durée de séjour dans l'atelier $Fi = C_i - r_i^2$.
- L'indicateur de retard de la tâche U_i : $\begin{cases} 1 \text{ si} & T_i \succ 0, \\ 0 \text{ sinon.} \end{cases}$

  On cherche alors à minimiser un ou plusieurs critères $F(C_1, ..., C_n)$.

  - Une fonction générique de **coût maximum** $f_{\max} = f_i(C_i)$.
    - La durée totale de l'ordonnancement $C_{\max} = \max_i C_i$.
    - Le retard algébrique maximum $L_{\max} = \max_i L_i$.
    - Le retard maximum $T_{\max} = \max_i T_i$.
  - Une fonction générique de **coût total** $\sum f = \sum_i f_i(C_i)$.
    - La somme (pondérée) des dates de fin $\sum_i w_i C_i$.
    - La somme (pondérée) des retards $\sum_i w_i T_i$.
    - Le nombre P (pondéré) des tâches en retard $\sum_i w_i U_i$.
    - La durée moyenne de séjour $\frac{1}{n}\sum_i F_i$.
    - La somme (pondérée) des avances et des retards $\sum_i (\alpha_i E_i + \beta_i T_i)$.

  (Dans notre travail on s'intéresse au cas où la fonction objective prend la forme $\sum_i w_i T_i$)

Critères réguliers

Définition 2.2.1 *Un **critère** $F(C_1, ..., C_n)$ est dit **régulier** si et seulement si F est une fonction croissante des dates de fin des tâches.*

Corollaire 2.2.1 *Les critères C_{max}, L_{max}, T_{max}, $\sum_i w_i C_i$, $\sum_i w_i T_i$, $\sum_i w_i U_i$, $\sum_i w_i F_i$ sont réguliers;*

Alors que le critère $\sum(\alpha_i E_i + \beta_i T_i)$ n'est pas régulier.

Notations

En ordonnancement d'atelier, il est pratique d'introduire une notation qui permet d'identifier de manière synthétique et précise le type de problème abordé qui proposés par (Graham et al, 1979) [5].

Un problème est décrit par un triplet $\alpha \mid \beta \mid \gamma$ où:

- $\alpha = \alpha_1\alpha_2$: environnement machine; α_1 représente le type et α_2 est le nombre de machines.

  - Le paramètre $\alpha_1 \in \{$ 1 ou $\emptyset, P, Q, R, O, F, J, FH \}$ caractérise le type de machines utilisées:

Valeur	Description
$\emptyset$	Machine unique
P	Machines parallèles identiques
Q	Machines parallèles uniformes
R	Machines parallèles quelconques
F	Système Flow shop
J	Système Job shop
O	Système Open shop
FH	Système Flow shop Hybride

Tableau 2.1: le type de paramètre α_1.

- $\alpha_2 \in \{\emptyset, k\}$ est un entier qui représente le nombre de machines dans le problème.

- $\alpha_2 = \emptyset$: le nombre de machines est supposé être variable.
- $\alpha_2 = k$: le nombre de machines est égal à k (k entier positif).

- β: Les caractéristiques des tâches, $\beta = \beta_1\beta_2\beta_3\beta_4\beta_5\beta_6\beta_7\beta_8$ décrit les tâches et les caractéristiques des ressources.

Valeur	Description
r_i	Une date de début au plus tôt r_i est associée à chaque tâche i
d_i	Une date d'échéance préférentielle d_i est associée à chaque tâche i
$prec$	Il existe des contraintes de précédence générale entre les opérations
$p_i = 1$	Les durées opératoires sont unitaires
$pmtn$	La préemption des opérations est autorisée
no_wait	Les opérations de chaque travail doivent se succéder sans attente
$Snsd(Rsnd)$	Les ressources doivent être préparées avant et/ ou après chaque exécution indépendamment de la séquence des tâches

Tableau 2.2: Les caractéristiques des tâches de paramètre β.

- γ: Le (ou les) critère(s) à optimiser.

Exemples

-1 $\| \sum w_i C_i$ représente le problème de minimisation de la somme pondérée des dates de fin dans le problème à une machine.

-$F3 \mid r_i \mid \sum T_i$ représente le problème de minimisation de la somme des retards dans un Flow-shop à trois machines avec contraintes de dates de disponibilité.

-$P_3 \mid pmtn, prec \mid L_{max}$ désigne le problème de minimiser le retard maximum de tâches liées par des contraintes de précédence arbitraires, à trois machines parallèles identiques où la préemption est autorisée.

2.3 Problème à une machine

L'étude des environnements à machine unique peut apparaître comme un cas d'école. Ceux-ci peuvent toute fois servir de cadre de base pour élaborer des raisonnements en environnements multi-machines, plus réalistes mais plus complexe. En pratique en effet, il peut s'avérer que les difficultés d'ordonnancement d'un atelier proviennent d'une machine dominante sur laquelle se concentrent les conflits, les attentes, etc.

Toute tâche J_i, $i \in \{1, 2, ..., n\}$ de durée p_i (le temps de traitement) s'exécute sur une machine qui ne peut traiter plus qu'une tâche à la fois.

Le champ α_1 est absent et $\alpha_2 = 1$.

Dans ce cas, l'ensemble des tâches à réaliser est fait par une seule machine. Les tâches alors sont composées d'une seule opération qui nécessite la même machine. L'une des situations intéressantes où on peut rencontrer ce genre de configurations est le cas où on est devant un système de production comprenant une machine glouton qui influence l'ensemble du processus. L'étude peut alors être restreinte à l'étude de cette machine.

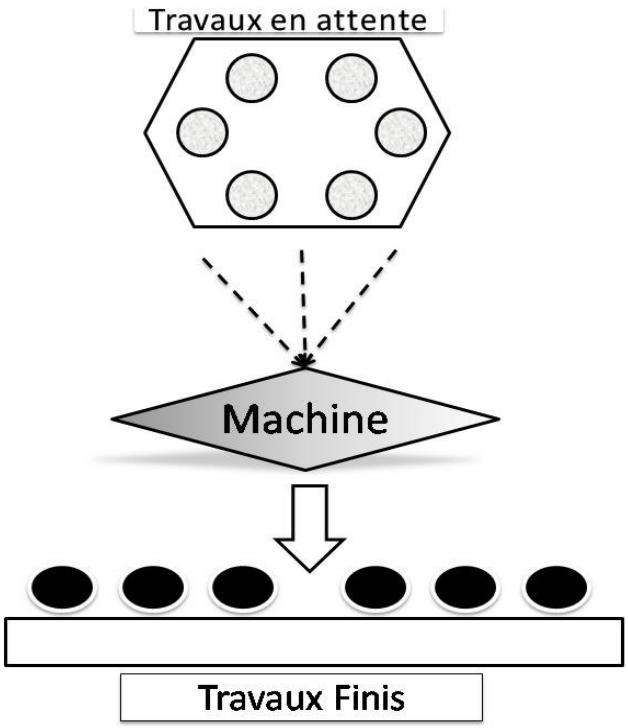

Figure 2.1: problème à une machine.

2.3.1 Hypothèses

- $m = 1$ et la machine est toujours disponible.
- Les n tâches comprennent chacun une seule opération à laquelle ils sont assimilés.
- Initialement, il n'existe pas de contrainte de précédence entre les différentes opérations.

- Les temps de préparation sont indépendants de la séquence des opérations et sont inclus dans leurs durées.

2.3.2 Résultats de base−Règles de priorité

Ce paragraphe traite des résultats spécifiques du problème à une machine. Des règles visent à ordonnancer les opérations selon un ordre de priorité imposé et permettent d'aboutir efficacement à une résolution optimale pour certains critères. Les opérations sont ordonnancées à partir de $t = 0$; à la date t correspondant à une étape donnée de la procédure, on choisit parmi les opérations prêtes (c'est-à-dire dont la date de début au plus tôt est au plus égale à t et dont les opérations précédentes sont achevées) celle de plus forte priorité. La priorité peut être liée au plus petit temps opératoire, à la plus petite date de fin au plus tard, à la plus grande quantité de travail restant à accomplir.

L'intérêt de ces méthodes réside dans leur simplicité d'application. En revanche, par leur caractère local, elles n'offrent aucune garantie d'optimalité.

2.3.3 Problème $1||\sum w_i C_i$

- Toutes les tâches sont disponibles à l'instant $t = 0$ $(\forall i, r_i = 0)$.
- w_i est le poids de la tâche J_i (ou tout simplement i).
- C_i la date de fin de i (c'est une inconnue).
- Un ordonnancement est complètement déterminé par une séquence des tâches.
- Minimiser $\sum w_i C_i$ revient à minimiser les encours totaux (pondérés)

tâche i	1	2	3	4	5
p_i	2	3	6	5	1
w_i	1	3	2	1	2

Exemple de séquence: $\pi_1 = (1, 2, 3, 4, 5)$

$\sum w_i C_i(\pi_1) = 1 * 2 + 3 * 5 + 2 * 11 + 1 * 16 + 2 * 17 = 89.$

Règle de Smith (SWPT):

En l'absence des contraintes d'antériorité, nous ordonnons les tâches dans l'ordre des ρ_i où $\rho_i = p_i/w_i$, croissants $\left(\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq ... \leq \frac{p_i}{w_i} \leq \frac{p_{i+1}}{w_{i+1}} \leq ... \leq \frac{p_n}{w_n}\right)$. Cette permutation est optimale [14].

Ordonnancer les tâches dans l'ordre croissant des p_i/w_i minimise le stock d'encours (i.e $\sum w_i C_i$).

tâche i	1	2	3	4	5
p_i	2	3	6	5	1
w_i	1	3	2	1	2
p_i/w_i	**2**	**1**	**3**	**5**	**1/2**

- La séquence optimale pour le problème avec poids est:

$\pi_w^* = (5, 2, 1, 3, 4)$.

- La séquence optimale pour le problème sans poids est:

$\pi_1^* = (5, 1, 2, 4, 3)$.

Théorème 2.3.1 *La règle SWPT donne une solution optimale pour* $1 \parallel \sum w_i C_i$ *en un temps polynomial.*

2.3.4 Problèmes $1|r_i|\sum w_i C_i$ et $1|r_i, pmtn|\sum w_i C_i$

Règle de Smith Préemptif

Affecter la machine à la tâche de plus courte durée sous la condition suivante: quand une tâche arrive, elle préempte la machine si sa durée est plus petite que la durée résiduelle de la tâche en cours [5].

Théorème 2.3.2 *La règle ci-dessus donne une solution optimale en un temps polynomial pour le problème* $1|r_i, pmtn|\sum C_i$ [7].

Théorème 2.3.3 *Les problèmes* $1|r_i|\sum w_i C_i$ *et* $1|r_i, pmtn \mid \sum w_i C_i$ *sont* **NP − difficile** [7].

2.3.5 Problèmes $1||L_{max}$ et $1|r_i, pmtn|L_{max}$

Règle de Jackson (EDD)

La règle de Jackson consiste à donner la priorité la plus élevée à la tâche disponible de plus petite date échue: $d_1 \leq d_2 \leq ... \leq d_i \leq d_{i+1} \leq ... \leq d_n$ [7].

Définition 2.3.1 *On désignera par ordonnancement de Jackson l'ordonnancement de liste obtenu en appliquant la règle de Jackson.*

Si $\forall i$, $r_i = 0$, *on parle également d'ordre EDD:* ***E****arliest* ***D****ue* ***D****ate.*

Théorème 2.3.4 *La règle EDD donne une solution optimale pour le problème* $1||L_{max}$ *en un temps polynomial* [8].

Règle de Jackson Préemptif

Affecter la machine à la tâche de plus petite date échue sous la condition suivante: quand une tâche arrive, elle préempte la machine si sa date échue est plus petite que la date échue de la tâche en cours [5].

Théorème 2.3.5 *La règle ci-dessus donne une solution optimale pour le problème* $1|r_i, pmtn|L_{max}$ *en un temps polynomial* [7].

Théorème 2.3.6 *Le problème* $1|r_i|L_{max}$ *est* $\mathbf{NP-difficile}$.

Exemple 2.3.1

i	1	2	3	4	5	6	7
r_i	0	9	13	11	24	30	36
p_i	6	7	6	7	4	3	5
d_i	31	41	22	24	27	40	48

- *Ordonnancement de Jackson préemptif*

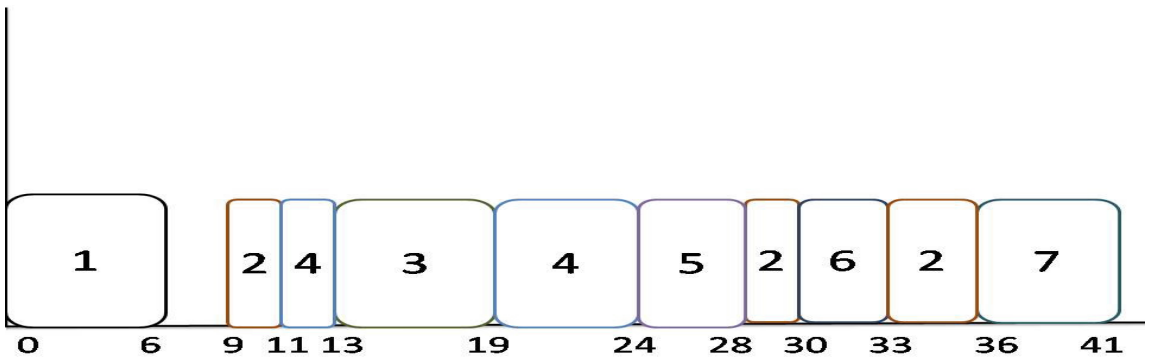

Figure 2.2: ordonnancement de Jackson préemptif.

- Ordonnancement de Jackson non-préemptif

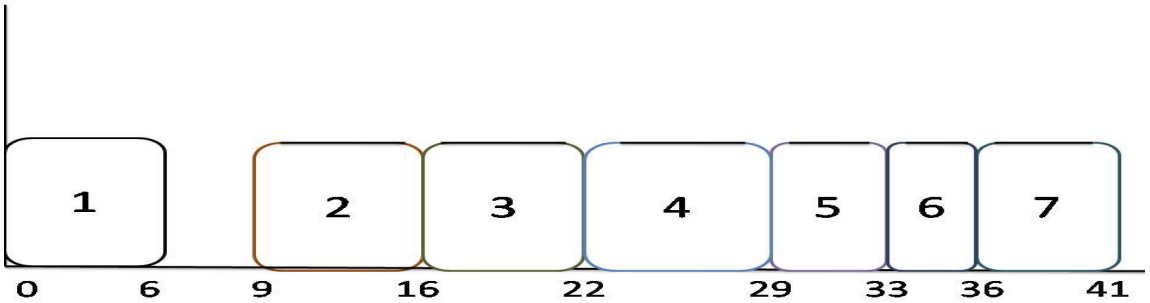

Figure 2.3: ordonnancement de Jackson non-préemptif.

2.3.6 Problème $1||\sum U_i$

On définit l'indicateur de retard de la tâche i par:

$$U_i = \begin{cases} 1 \text{ si } C_i \succ d_i \\ 0 \text{ sinon.} \end{cases}$$

Alors: $\sum_{i=1}^{n} U_i$ désigne le nombre de tâches en retard.

Une Considéré l'ordonnancement dite EDD où les n tâches sont ordonnancer telles que $d_1 \leq d_2 \leq ... \leq d_n$.

Règle de Hodgson

Placer les tâches, une par une, dans l'ordre des dates échues croissantes, en rejetant la tâche déjà placée de plus grande durée à la fin de l'ordonnancement si un retard apparaît.

Théorème 2.3.7 *La règle ci-dessus donne une solution optimale pour le problème* $1||\sum U_i$ *en un temps polynomial* [7].

Théorème 2.3.8 *Le problème* $1||\sum w_i U_i$ *est* **NP − difficile** [4].

NB: La règle de Hodgson n'est pas optimal pour $1||\sum w_i U_i$.

2.4 Problèmes à machines parallèles

Dans ce cas, on dispose d'un ensemble de machines identiques pour réaliser les tâches. Les tâches se composent d'une seule opération et un travail exige une seule machine. L'ordonnancement s'effectue en deux phases: la première phase consiste à affecter les travaux aux machines et la deuxième phase consiste à établir la séquence de réalisation sur chaque machine.

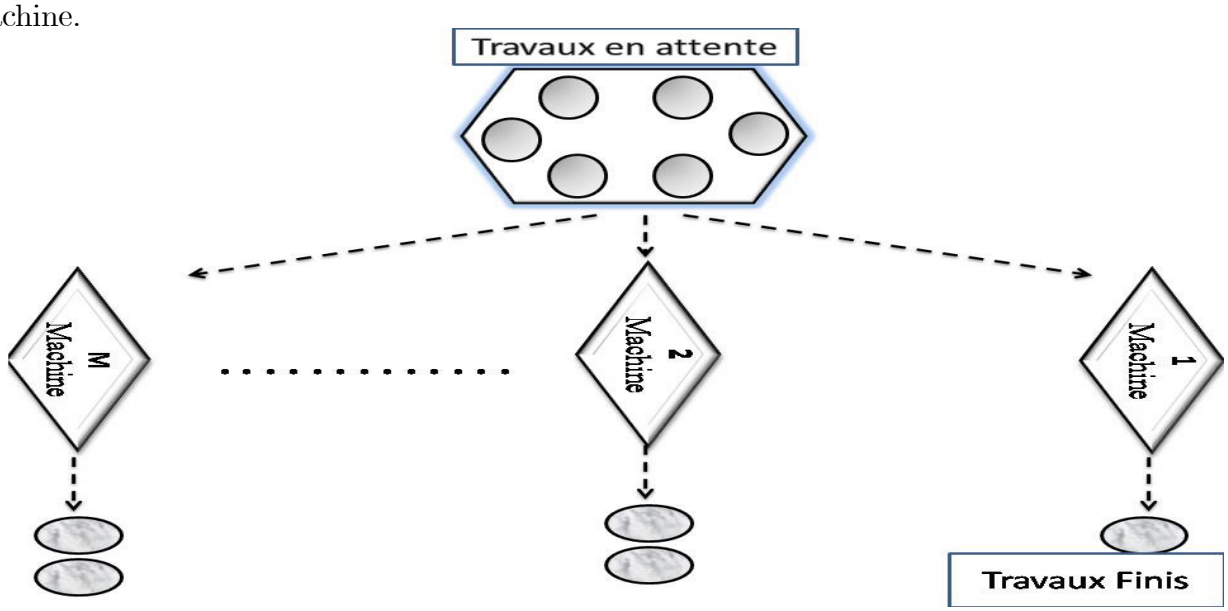

Figure 2.4: problème à machines parallèles.

- $p_{i,j}$ est la durée d'exécution de J_i sur la machine M_j, $j = 1, 2, ...m$.

- Si $\alpha_1 = P$ alors **machines identiques** $\Rightarrow \forall\ j,\ p_{i,j} = p_i$.
- Si $\alpha_1 = Q$ alors **machines uniformes** $\Rightarrow \forall j,\ p_{i,j} = p_i/s_j$ où s_j est la vitesse de traitement M_j.
- Si $\alpha_1 = R$ alors **machines indépendantes** $\Rightarrow \forall j,\ p_{i,j} = p_i/s_{i,j}$ où $s_{i,j}$ est la vitesse de traitement de la tâche J_i par la machine M_j.
- Si α_2 est un entier positif, le nombre de machines est supposé constant. Si α_2 est absent alors ce nombre est supposé arbitraire.
- Deux types de décision à prendre
  - Affectation des tâches aux machines.
  - Séquencement de chaque machine:
    - D'une façon indépendante s'il n'y a pas de contraintes de précédence entre les tâches.
    - D'une "façon dépendante" sinon.

2.4.1 Problème $P \mid (p_i = p) \mid C_{max}$

Théorème 2.4.1 *Un solution optimale pour le problème $P \mid (p_i = p) \mid C_{max}$ est déterminer par:*

- Ordonnancer les tâches de 1 à n sur les machines de 1 à m par ordre.

Cette algorithme donne une solution optimale au problème considérer. Elle est d'ordre $O(n\ log\ n)$ où n = nbre des tâches.

Preuve. Par construction $C_{\max} = \lceil \frac{\sum p_i}{m} \rceil$ qui est la plus petite valeur possible que prend $C_{\max}$. ■

NB: L'algorithme ci-dessus ne donne pas une solution optimale pour le problème $P \parallel C_{\max}$ où les temps de traitement des tâches sont différents.

Remarque 2.4.1 *Le problème $P||C_{max}$ est* **NP − difficile** [10].

2.4.2 Problème $P \mid (p_i = p) \mid \sum w_i C_i$

Théorème 2.4.2 *Une solution optimale pour le problème* $P \mid (p_i = p) \mid \sum w_i C_i$ *est déterminer par:*

- *Ordonnancer les tâches selon* $w_i \prec w_j \Longrightarrow i \prec j$ *pour déterminer* $(i_1, i_2, ..., i_n)$.
- *Affecter les tâches* i_1 *à* i_n *sur les machines de* 1 *à* m.

Preuve. La justification de cet algorithme est basée sur la règle de $SWPT$ de Smith.

■

Remarque 2.4.2 *Le problème* $P||\sum C_i$ *est* **NP − difficile**.

2.5 Classification des ateliers

Une classification des problèmes d'ordonnancement dans un atelier peut s'opérer selon le nombre de machines et leurs ordres d'utilisation pour fabriquer un produit, qui dépend de la nature de l'atelier considéré. Un atelier est caractérisé par le nombre de machines qu'il contient et par son type.

Comme le montre **la figure 1.1**, on distingue les trois types d'ateliers suivants:

- Ateliers à cheminement unique (Flow-shop).
- Ateliers à cheminements multiples (Job-shop).
- Ateliers à cheminements libres (Open-shop).

- m machines différentes M_j, $j \in \{1, ..., m\}$.
- n tâches J_i, $i \in \{1, ..., n\}$.
  - Chaque tâche J_i est décrit par n_i tâches ou opérations $O_{i,j}, j \in \{1, ..., n_i\}$.
  - La durée d'une opération $O_{i,j}$ est $p_{i,j}$.
  - La machine qui exécute l'opération $O_{i,j}$ du tâche est notée $M(O_{i,j})$ ou $M_{i,j.}$
  - Les opérations d'une même tâche ne peuvent pas être exécutées simultanément**.**
- α_2 (voir machines parallèles).

2.5.1 Les ateliers à cheminement unique (Flow-shop)

Un atelier à cheminement unique est un atelier où le processus d'élaboration de produits est dit « linéaire », c'est-à-dire lorsque les étapes de transformation sont identiques pour tous les produits fabriqués. Selon les types de produits élaborés, on distingue la production continue et la production discrète. La production continue est caractérisée par la fluidité de son processus et l'élimination du stockage. C'est le cas notamment dans les raffineries, les cimenteries, les papeteries. La production discrète de masse s'applique principalement aux produits de grande consommation fabriqués à la chaîne.

- Chaque tâche est constitué de m opérations et l'ordre de passage sur les différentes machines est le même pour tous les tâches J_i: $O_{i,1} \rightarrow O_{i,2} \rightarrow, ..., \rightarrow O_{i,m}$ et $M_{i,j} = M_j$.
- $\alpha_1 = F$

L'un des objectifs principaux dans le cas d'atelier à cheminement unique est de trouver une séquence des tâches en main qui respecte un ensemble de contraintes et qui minimise le temps total de production. Parmi les caractéristiques d'un problème de cette catégorie:

- Il existe au minimum $n!$ différentes solutions où n est le nombre de tâches à réaliser. Notons que $n! = n * (n-1) * (n-2) \ldots . * 1$;

- Le problème est $NP - difficile$ à l'exception des versions avec deux machines et certains cas particuliers avec trois machines;

- Une grande productivité mais une faible $flexibilité$.

Présentation des ateliers de type Flow-shop

Un problème d'ordonnancement Flow-shop met en œuvre m machines, notées $M_1, M_2, ..., M_m$, et n produits (tâches), notées $p_1, p_2, ..., p_n$. Chaque produit (tâche) est exécuté(e) sur la machine M_1, puis sur M_2, et ainsi de suite, jusqu'à ce qu'à la dernière machine. A chaque instant, une machine traite au plus un produit à la fois et chaque produit est exécuté par au plus une machine.

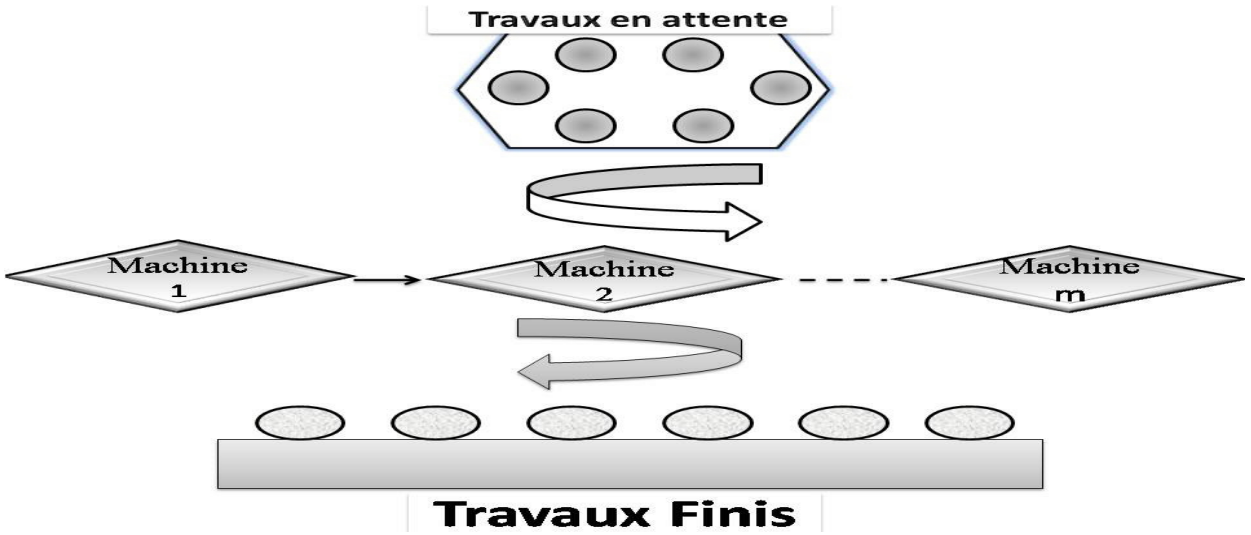

Figure 2.5: cheminement des produits dans un atelier de type Flow-shop.

Problème F2$||C_{max}$

- n tâches $J_i, i = 1, ..., n$, à exécuter sur deux machines M_1 et M_2.
- Cheminement unique: M_1 puis M_2.
- a_i et b_i sont les durées de la tâche $J_i, i = 1, ..., n$ sur les machines M_1 et M_2, respectivement.
- Intuitivement
  - Avoir des tâches exécutées le plus rapidement possible sur M_1 pour pouvoir les exécuter M_2(M_2 reste inactive le minimum de temps possible)

  $\Longrightarrow$ Règle **SWPT** sur M_1.

  - Sur M_2, on essaie d'exécuter les tâches dans l'ordre LPT pour que M_2 n'attende pas trop que M_1 lui fournisse des tâches.

Définition 2.5.1 *Un ordonnancement pour lequel toutes les tâches sont exécutées dans un même ordre (sur les machines) est appelé* ***ordonnancement de permutation.***

Lemme 2.5.1 *Une instance du problème $F2||C_{max}$ a toujours une solution optimale qui est un ordonnancement de permutation.*

Théorème 2.5.1 *La règle de Johnson est optimale pour* $F2||C_{max}$ [5].

2.5.2 Les ateliers à cheminements quelconques (Job-shop)

- Le nombre d'opérations n'est pas forcément le même pour tous les tâches.
- Chaque tâche a son propre ordre de passage sur les machines.
- $\alpha_1 = J$.

Ce sont des ateliers où les opérations sont réalisées selon un ordre bien déterminé, variant selon la tâche à exécuter; le Job-shop flexible est une extension du modèle Job-shop classique; sa particularité réside dans le fait que plusieurs machines sont potentiellement capables de réaliser un sous-ensemble d'opérations.

L'objectif le plus considéré dans le cas d'un atelier à cheminements multiples est le même que celui considéré pour un atelier à cheminement unique, à savoir trouver une séquence de tâches sur les machines qui minimise le temps total de production.

Présentation des ateliers de type Job-shop

Les données: les données du problème sont [6][15]:

Un ensemble M de m machines: une machine est notée M_j avec $j = 1, ..., m$. Chaque machine ne peut effectuer qu'un seul type d'opérations. Le nombre total d'opérations exécutées par cette machine est noté m_j.

Un ensemble J de n tâches: un tâches est noté J_i avec $i = 1, ..., n$. Chaque tâche est composé d'une gamme opératoire, i.e. une séquence linéaire fixée de n_i opérations. Cette séquence ne dépend que du tâche, et peut varier d'un tâche à l'autre.

L'opération $O_{i,j}$ est la $Jeme$ opération dans la gamme opératoire de J_i. Elle nécessite l'utilisation de la machine M_j. Le fait que la machine demandée par l'opération $O_{i,j}$.

La figure suivante montre un exemple d'un atelier à cheminements multiples avec quatre travaux et six machines.

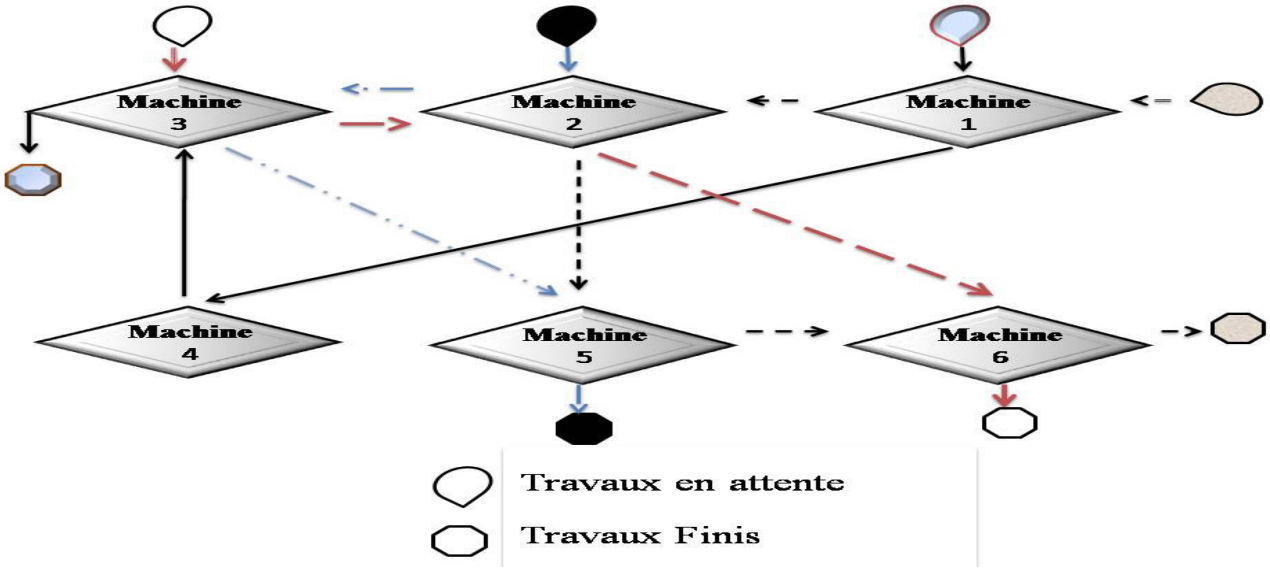

Figure 2.6: cheminement des produits dans un atelier de type Job-shop.

J_i	j_1			j_2			j_3	
$O_{i,j}$	$O_{1,1}$	$O_{1,2}$	$O_{1,3}$	$O_{2,1}$	$O_{2,2}$	$O_{3,2}$	$O_{3,1}$	$O_{3,2}$
$M_{i,j}$	M_1	M_2	M_3	M_2	M_1	M_3	M_3	M_2
$p_{i,j}$	3	2	5	4	2	2	21	3

2.5.3 Les ateliers à cheminements libres (Open-shop)

- Le nombre d'opérations n'est pas forcément le même pour tous les tâches.
- L'ordre de passage sur les machines est totalement libre.
- $\alpha_1 = O$.

Ce type d'atelier est moins contraint que celui de type Flow-shop ou de type Job-shop. Ainsi, l'ordre des opérations n'est pas fixé a priori; le problème d'ordonnancement consiste, d'une part, à déterminer le cheminement de chaque produit et, d'autre part, à ordonnancer les produits en tenant compte des gammes trouvées, ces deux problèmes pouvant être résolus simultanément. Comparé aux autres modèles d'ateliers, l'open-shop n'est pas couramment utilisé dans les entreprises.

On peut remarquer que le problème de l'open-shop peut être interprété comme un problème à tâches indépendantes dans lequel chaque tâche nécessite simultanément deux

ressources: la tâche et la machine. Ces deux ressources introduisent chacune des contraintes disjonctives. Il en résulte une certaine symétrie entre les travaux et les machines, comme cela apparaît par exemple ci-après pour l'interruption des tâches.

2.6 Conclusion

Dans ce chapitre nous avons traité le problème d'ordonnancement dans les différents types d'atelier et l'utilisation des règles de priorité dans l'ordonnancement des différents ateliers. La première partie de ce chapitre, a été consacrée aux règles de priorités où nous avons défini ces règles et donné les principaux critères de leurs évaluations puis leurs classifications, nous avons abordé aussi la combinaison des règles de priorité, son intérêt et les limites de cette combinaison. La seconde partie visait à situer l'ordonnancement dans les différents types d'atelier: Flow-shop, Job-shop et Open shop.

Chapitre 3

Etude de cas

3.1 Introduction

Dans ce chapitre on considère une étude de cas qui est l'application de deux heuristiques $SWPT$ (Shortest Weighted Processing Time qui est optimale pour $1 \parallel \sum w_i C_i$) et EDD (Earliest Due Date qui est solution optimale pour $1 \parallel L_{max}$) pour déterminer une solution approchée pour le problème d'ordonnancement de n tâches sur une machine unique afin de minimiser la somme pondéré des retards($1 \parallel \sum w_i T_i$). Dans la premier section on donne l'algorithme général qui en globales les deux heuristiques. Alors que dans la deuxième section on donne une simulation empirique : la façon de génération des données, comparaison d'implémentation.

Dans la dernière section on donne une conclusion générale.

3.2 Solution exacte pour des problèmes polynomiaux

3.2.1 Solution optimale pour ($1 \parallel \sum w_i C_i / 1 \parallel L_{\max}$)

Etape 1:

$j = \{1, ..., n\}$

$f = 0, l = 1$

$C = 0$

Etape 2:

Tant que $j \neq \emptyset$ faire

Déterminer la tâche de meilleur index

Etape 3:

Ordonnancer la tâche h sur la machine

Poser $\delta(l) = h, J = J/\{h\}, C = C + p_h$

$f = f + w_h C$

$l = l + 1$

Aller en **étape 1**

Algorithme 3.1: solution optimale pour $1 \parallel \sum w_i C_i \ / 1 \parallel L_{max}$.

Remarque 3.2.1 - *Index pour SWPT*$\rightarrow \min \frac{p}{w}$

- *Index pour EDD*$\rightarrow \min d$.

3.3 Heuristiques pour $1 \parallel \sum w_i T_i$

Algorithme pour heuristiques(1,2)

Etape 1:

$J = \{1, ..., n\}$

$f = 0, l = 1$

$C = 0, L = 0, T = 0.$

Etape 2:

Tant que $J \neq \emptyset$ faire

Déterminer la tâche de meilleur index

Etape 3:

Ordonnancer la tâche h sur la machine

Poser $\delta(l) = h, J = J/\{h\}, C = C + p_h$

Etape 4:

$L = C - d_h$

$T = max\{0, L\}$

$f = f + w_h T$

$l = l + 1$

aller en **étape 2**

Algorithme 3.2: algorithme pour heuristiques(1,2).

Remarque 3.3.1 - *Index pour SWPT*$\rightarrow \min \frac{p}{w}$

- *Index pour EDD*$\rightarrow \min d$.

Méthode approché pour $P \parallel \sum w_i T_i$

Etape 1:

$J = \{1, ..., n\}, j = \emptyset$

$f = 0, l = 1$

$j = 1, ..., m$

$C_j = 0, L_j = 0, T_j = 0.$

Etape 2:

Tant que $J \neq \emptyset$ faire

Déterminer l'ensemble $h \in J$ telque

$\frac{p_h}{w_h} = \min_{k \in J}\{\frac{p_k}{w_k}\}$

Fin

Etape 3:

Déterminer la machine $j \in \{1, ..., m\}$ telque

$C_j = \min\{C_\alpha\}_{\alpha=1,m}$

Ordonnancer la tâche h sur la machine j

Poser $\delta(l) = h, J = J/\{h\}, C_j = C_j + p_h$

Etape 4:

$L_j = C_j - d_h$

$T_j = max\{0, L_j\}$

$f = f + w_h T_j$

$l = l + 1$

aller en **étape 2**

Algorithme 3.3: méthode approché pour $P \parallel \sum w_i T_i$.

3.4 Simulation empirique

3.4.1 Générations des données

Notre problème consiste à minimiser la somme pondérée des retards des tâches. Pour étudier ce problème nous avons adopté les deux règles: $SWPT$ et EDD. Pour générer les instances du test nous avons pris les valeurs des poids $w_i \in [1, 10]$; les valeurs du temps d'exécution $p_i \in [1, 200]$ et date d'achèvement souhaitée de la tâche $d_i \in [100, 4000]$.

Une série de résultats numériques est présentée dans cette section. Elle nous permet de comparer l'efficacité de chacune de ces sommes (pondérées) des retards. Cette étude est composée de quatre parties: pour $n = 5, 10, 20$ et 30 tâches. On a généré 10 problèmes pour chaque instante n. Sur chacun des tableaux présentés dans cette section, nous rapportons le minimum de valeur de la fonction objectif $\sum w_i T_i$ en utilisant deux Règles $SWPT$ et EDD.

3.4.2 Résultats numériques

En gras est la meilleur valeur de $\sum w_i T_i$.

1. Pour $n = 5$

   Dans exemple 1:

Problème	1	2	3	4	5	6	7	8	9	10
$SWPT$	220	160	85	1210	**820**	**128**	155	2118	**157**	706
EDD	**160**	**10**	**25**	**1080**	1490	400	**120**	**1197**	376	**206**

Tableaux 3.1: résultat de calcul pour $n = 5$ pour chacun 10 problèmes.

2. Pour $n = 10$

   Dans exemple 2:

Problème	1	2	3	4	5	6	7	8	9	10
$SWPT$	1721	**2183**	14926	**8564**	**7257**	**7739**	3462	**11634**	**2321**	4139
EDD	**1106**	2510	**14613**	15059	11458	14786	**445**	26223	7336	**2697**

Tableaux 3.2: résultat de calcul pour $n = 10$ pour chacun 10 problèmes.

3. Pour $n = 20$

   Dans exemple 3:

Problème	1	2	3	4	5	6	7	8	9	10
$SWPT$	**16065**	**11598**	17310	**18234**	**17397**	10490	**6999**	3351	**4455**	**5613**
EDD	37634	29012	**16113**	38740	34806	**5589**	11230	**1089**	23545	22355

Tableaux 3.3: résultat de calcul pour $n = 20$ pour chacun 10 problèmes.

4. Pour $n = 30$

   Dans exemple 4:

Problème	1	2	3	4	5	6	7	8	9	10
$SWPT$	**42883**	**28152**	**34264**	**72324**	12349	**43960**	**44157**	**48521**	**25478**	**28266**
EDD	114068	78582	69609	139357	**4209**	127841	86834	160447	76892	46781

Tableaux 3.4: résultat de calcul pour $n = 30$ pour chacun 10 problèmes.

n	$SWPT$	EDD
5	30%	70%
10	60%	40%
20	70%	30%
30	90%	10%

Tableaux 3.5: comparaison des résultats

On remarque que $SWPT$ performe mieux quand n augmente contrairement à la règle EDD.

3.4.3 Discussion des résultats

On remarque des fois que la déférence est très significative entre les valeurs de la fonction objective de l'ordonnancement trouvé par les deux méthodes. Ceci pourrait être interpréter par le fait que les d_i peuvent être dispersées. Par conséquent les différences entre les dates de fin et les dates échues deviennent petites. Comme résultat les retards deviennent petits et ainsi la valeur de la fonction objectif.

3.5 Plateforme de travail

L'algorithme d'affectation des tâches a été programmé par **Java** et la disposition des tâches ressources par ms-Project. Pour évaluer ce programme, nous avons considéré un ensemble de n tâches préventives à ordonnancer en fonction du temps d'exécution et des ressources.

Java

Le langage Java est un langage de programmation informatique orienté objet créé par *James Gosling* et *Patrick Naughton*, employés de Sun Microsystems, avec le soutien de *Bill Joy* (cofondateur de Sun Microsystems en 1982), présenté officiellement le 23 *mai* 1995 au *SunWorld*.

La société Sun a été ensuite rachetée en 2009 par la société Oracle qui détient et maintient désormais *Java*.

La particularité et l'objectif central de Java est que les logiciels écrits dans ce langage doivent être très facilement portables sur plusieurs systèmes d'exploitation tels que $UNIX, Windows, Mac\ OS$ ou $GNU/Linux$, avec peu ou pas de modifications. Pour cela, divers plateformes et frameworks associés visent à guider, sinon garantir, cette portabilité des applications développées en *Java*.

Java est un langage de programmation très utilisé, notamment par un grand nombre de développeurs professionnels, ce qui en fait un langage incontournable actuellement [9].

NetBeans

NetBeans est un environnement open-source de développement intégré pour développer en $Java, PHP, C++$, et d'autres langages de programmation. NetBeans est aussi appelé une plate-forme de composants modulaires utilisées pour développer des applications Java Desktop**.**

3.6 Conclusion

Dans ce chapitre; une étude de cas a été présenté. C'est en fait une étude comparative numérique entre deux heuristiques pour déterminer une solution approchée au problème d'ordonnancement de tâches sur une machine afin de minimiser la somme pondérée des retards. Ces heuristiques $H_1(SWPT); H_2(EDD)$ sont basées sur les solutions optimales de $1 \parallel \sum w_i C_i$ et $1 \parallel L_{\max}$. Les algorithmes décrivant ces heuristiques sont établis et des exemples numériques illustratifs sont donnés. De toutes les manières c'est une étude préliminaire qui a été faite; on a généré 40 problèmes d'ordonnancement et on a remarqué la supériorité de H_1 sur H_2 chaque fois que n augmenté.

Conclusion

Dans ce mémoire on a considéré une classe célèbre de problèmes d'optimisation combinatoire. Ceux sont les problèmes d'ordonnancement d'atelier. En fait on a pris deux catégories que se soit pour les problèmes d'ordonnancement de tâches sur une seule ou plusieurs machines identiques parallèles ou ceux de Job-shop, Flow-shop ou Open-shop. Et puisque l'étude des problèmes simples de base (polynomiaux) a une importance majeure pour l'étude des problèmes complexes ($NP-difficiles$) on s'est concentré à l'étude des premiers problèmes, i.e. ceux qu'on a pu déterminer un algorithme de complexité polynomiale pour leurs solutions.

Citons les problèmes a ressource unique $1 \parallel \sum C_i$, $1 \parallel \sum F_i$, $1 \parallel \sum w_i C_i$, $1 \parallel \sum w_i F_i$, $1 \parallel L_{\max}$, $1 \mid (p_i = p) \mid C_{\max}, 1 \parallel \sum U_i$. Alors pour les problèmes à ressource multiple citons le problème $F_2 \parallel C_{\max}$.

Les solutions de ceux-ci peuvent être utilisées comme heuristiques pour déterminer une solution admissible pour les mêmes problèmes avec d'autres contraintes supplémentaires. Par conséquent les valeurs des solutions exactes trouvées peuvent être considérées comme solutions approchées pour les problèmes complexes. Il est important de noter aussi que la valeur de la solution optimale pour un problème polynomial pourrait être servie comme évaluation par défaut (excès) sur la fonction objective d'un problème $NP\ difficile$ déduit de premier en imposant d'autres contraintes.

En fait et comme étude de cas on a considéré le problème $1 \parallel \sum w_i T_i$ qui est $NP\ difficile$ et on a essayé d'appliquer des algorithmes (heuristiques) telle que $SWPT$, EDD (optimales respectivement pour $1 \parallel \sum w_i C_i, 1 \parallel L_{\max}$).

Une étude numérique comparative entre la performance de ces méthodes a été faite en utilisant le langage $Java$.

Bibliographie

[1] B. Cardoen, E. Demeulemeester & J. Belien. Operating room planning and scheduling. Revue European Journal of Operational Research, n° 201, (2010), 921-932.

[2] J. Carlier et P. Chrétienne. Problèmes d'ordonnancement, Modélisation, Complexité, Algorithmes. Edition Masson, Paris, (1988).

[3] F.T.S. Chan. Resource management in project scheduling through simulation. International Journal of Computer Applications in Technology, vol 10, n° 1 / 2, (1997), 81 – 89.

[4] J. Du, J.Y.T. Leung. Minimizing total tardiness on one machine is N.P. (1990).

[5] P. Esquirol, P. Lopez. L'ordonnancement. Economica, (1999).

[6] G. Fink. Recherche opérationnelle et réseaux. Lavoisier, Paris, 2002.

[7] S. French. Sequencing and scheduling: An introduction to the mathematics of the Job-shop. Wiley, New York, (1982).

[8] S.M. Johnson. Optimal two- and three- stage production schedules with setup times included. Naval Research Logistics Quarterly, (1954).

[9] C. Herby. Apprenez à programmer en Java. La programmation professionnelle à la portée de tous. Imprim'vert, France, (2011).

[10] R. Karp, R. E. Miller and J.W. Thatcher (eds). Complexity of computer computations. Plenum Press, New-York (1972).

[11] A. Letouzey. Ordonnancement interactif basé sur des indicateurs: Applications à la gestion de commandes incertaines et à l'affectation des opérateurs. Thèse de Doctorat. Institut National Polytechnique de Toulouse, (2001).

[12] R. M. V. Rachamadugu, Thomas & E. Morton. Myopic Heuristics for the weighted tardiness problem on identical parallel machines. The robotics institute. Carnegie Mellon University, Pittsburgh, 1983.

[13] E. Saule, P-F. Dutot and G. Mounie. Scheduling With Storage Constraints. In Electronic proceedings of IPDPS 2008. Miami, Florida USA, APR 2008.

[14] W. E. Smith. Various optimizers for single state production. Naval Research Logistic Quarterly, n°3, (1956), 59-66.

[15] J. PH. Vacher. Un système adaptatif par agents avec utilisation des algorithmes génétiques multi-objectifs: Application à l'ordonnancement d'atelier de type Job-shop N×M. Thèse de Doctorat, Université du Havre, (2000).

[16] M.B. Wall. A Genetic Algorithm for Resource Constrained Scheduling. Ph. D thesis, Massachusetts Institute of Technology, (1996).